

# Princeton University

## COS 217: Introduction to Programming Systems

### Emacs Tutorial and Reference

## Part 1: Tutorial

This tutorial describes how to use a minimal subset of the `emacs` editor. See the `emacs` reference in Part 2 of this document for more information. Also see the *GNU Emacs Tutorial* document at <http://www.gnu.org/software/emacs/tour>. Throughout the tutorial text in **boldface** indicates hands-on activities.

The tutorial assumes that you've copied the file `/u/cos217/.emacs` to your home directory, as described in the *A Minimal COS 217 Computing Environment* handout from our first precept. It also assumes that you've copied files named `hello.c`, `circle.c`, and `testintmath.c` into your working directory. (Those files contain C programs that we'll describe in upcoming precepts.) Those files are available in the directory `/u/cos217/emacsstestfiles`. You can issue this command to copy them to your working directory:

```
cp /u/cos217/emacsstestfiles/* .
```

## Background

The `emacs` editor was created in the mid 1970s by Richard Stallman. Originally it was a set of "editing macros" for an editor that now is extinct.

The `emacs` editor is popular, for a few reasons. It is:

- Free. It's a component of the GNU tool set from the Free Software Foundation.
- Highly customizable. Emacs is written in the LISP programming language, and is easy to customize via that language.
- Integrated with other GNU software. In particular, `emacs` is integrated with the Bash history mechanism. Essentially you can think of the Bash history list as a "file"; you can use Emacs commands to scroll through and edit that file, and thereby easily reissue previous commands or variants thereof. Emacs also is integrated with the `gcc` compiler driver, as this tutorial describes. Finally, and probably most importantly, Emacs is integrated with the `gdb` debugger. A future precept will describe that integration.

The `emacs` editor is *modal*. That is, at any given time, `emacs` is in one of several modes. In the COS 217 course you will use *C mode*, *Assembler mode*, *Text mode*, and *Fundamental mode*. Emacs determines its mode based upon filename extensions. If the current file has a name whose extension is `.c`, then `emacs` is in C mode. If the current file has a name whose extension is `.s`, then `emacs` is in Assembler mode. If the current file has a name whose extension is `.txt`, then `emacs` is in Text mode. If the current file has a name which does not have an extension, then `emacs` is in Fundamental mode. This tutorial uses C mode.

## Launching emacs

To launch `emacs`, issue the `emacs` command followed by the name of the file that you wish to create or edit. For example, **issue this command at the Bash prompt:**

```
emacs testintmath.c
```

`emacs` loads the contents of the `testintmath.c` into a buffer in memory, and displays that buffer in the window. It places the point over the first character in the first line of the buffer.

Note the `emacs` terminology: A *buffer* is an area of memory. A *window* is a graphical entity that displays the contents of a specified buffer. The *point* is a small gray box which overlays a character, thus indicating which character is the "current" character.

## Notation

Throughout this document:

- `Esc somechar` means "type the `Esc` key followed by the *somechar* key."
- `Ctrl-somechar` means "type the *somechar* key while holding down the `Ctrl` key."

for any character *somechar*.

Incidentally, on a PC `Alt-somechar` (that is, typing the *somechar* key while holding down the `Alt` key) has the same effect in `emacs` as `ESC somechar` does. On a Mac that's true if and only if you have configured your Terminal application appropriately. To do that, from the Terminal application's menu choose `Terminal | Preferences... | Settings | Keyboard` and make sure the "Use option as meta key" checkbox is checked.

## The `.emacs` File

When you launch `emacs`, it looks for a file named `.emacs` in your home directory. If `emacs` finds that file, it assumes that the file contains configuration function calls, and executes them.

Take a look at the `.emacs` file that you copied to your home directory. **Issue the command `cat .emacs`** to do that. The file is thoroughly commented; please study it at your leisure. In particular, note this line:

```
(setq c-default-style "ellemtel")
```

As described below, `emacs` automatically indents your C code according to whatever indentation style you specify. That line sets the indentation style to `ellemtel`. The commented-out lines that immediately follow in the `.emacs` file show the names of some other styles. Any of those styles is fine in the context of the COS 217 course. Experiment! See which you like best.

## Calling Functions

In `emacs`, all work is accomplished by calling functions. The syntax for calling a function is:

```
Esc x function
```

For example, the `forward-char` function moves the point forward one character:

```
Esc x forward-char
```

`emacs` moves the point forward one character within the buffer each time you call the `forward-char` function. **Call `forward-char` a few times.**

Clearly there must be a better way to move the point! More generally, there must be a better way to call often-used functions.

## Key Bindings

There indeed is a better way. The most often-used functions are bound to keystrokes. For example, the `forward-char` function is bound to the keystroke `Ctrl-f`. **Type `Ctrl-f` a few times.** The `forward-char` function also is bound to the right-arrow key. **Type the right-arrow key a few times.**

Many keystrokes are bound by default. You also can bind your own, typically by placing a function call of this form in your `.emacs` file:

```
(global-set-key keystrokes 'function)
```

But few new emacs users create their own keystroke bindings.

## Moving the Point

The simplest way to move the point is via the `forward-char`, `backward-char`, `next-line` and `previous-line` functions, each of which is bound to an arrow key. **Type the arrow keys to move the point right, left, down, and up several times.**

The `beginning-of-line` and `end-of-line` functions have intuitive meanings. They are bound to the `Ctrl-a` and `Ctrl-e` keystrokes, respectively. They may also be bound to the `Home` and `End` keys, respectively; but `Home` and `End` may or may not work with your terminal emulation software. **Type `Ctrl-a`, `Ctrl-e`, `Home`, and `End` several times.**

Perhaps counter-intuitively, the `scroll-up` function moves the window downward in the buffer; equivalently, it moves the buffer upward in the window. The `scroll-up` function is bound to `Ctrl-v`, and also may be bound to the `PageDn` key. The `scroll-down` function moves the window upward in the buffer. That is, it moves the buffer downward in the window. The `scroll-down` function is bound to `ESC v`, and also may be bound to the `PageUp` key. **Type `Ctrl-v`, `PageDn`, `ESC v`, and `PageUp` several times.**

The `end-of-buffer` function moves the point to the end of the buffer; it is bound to `Esc >`. The `beginning-of-buffer` function moves the point to the beginning of the buffer; it is bound to the `Esc <`. **Type `Esc >` and `Esc <` several times.**

The `goto-line` function allows you to specify, by number, the line to which the point should be moved. It is bound to the `Ctrl-x 1` (that's `Ctrl-x` followed by the "ell" key) keystroke sequence. **Type `Ctrl-x 1`, followed by some reasonable line number, followed by the `Enter` key.**

## Inserting and Deleting

To insert a character, move the point to the character before which the insertion should occur, and then type the character. **Move the point to some arbitrary spot in the buffer, and type some characters.**

The `c-electric-backspace` function (bound to the `Backspace` key) deletes the character before the point. **Move the point to some arbitrary spot in the buffer, and type `Backspace` several times.** The `c-electric-delete-forward` function (bound to `Ctrl-d`) deletes the character at the point. **Move the point to some arbitrary spot in the buffer, and type `Ctrl-d` several times.**

To delete a line, move the point to the beginning of the line and then call the `kill-line` function (bound to `Ctrl-k`). Calling the function once kills the characters comprising the line, but not the line's end-of-

line mark. Calling the function a second time also kills the end-of-line mark. **Move the point to the beginning of some arbitrary line, and type `Ctrl-k` several times.**

Actually, the `kill-line` function doesn't completely discard the line that it kills; instead it moves the line to the `emacs` clipboard. The `yank` function (bound to `Ctrl-y`) copies (*yanks*) the line from the `emacs` clipboard into the buffer at the point. The combination of the `kill-line` and `yank` functions provides a single-line cut-and-paste functionality, as this sequence illustrates:

- **Move the point to the beginning of some non-empty line that you wish to move.**
- **Type `Ctrl-k` twice.**
- **Move the point.**
- **Type `Ctrl-y`.**

For multiple-line cut-and-paste, you must know about `emacs` *regions*. A region is an area of text that is bounded by the point and the *mark*. The `set-mark-command` function (bound to `Ctrl-Space`) sets the mark. The `kill-region` function (bound to `Ctrl-w`) moves the region to the `emacs` clipboard; effectively it wipes out the region. This sequence illustrates moving multiple contiguous lines from one place to another in the buffer:

- **Move the point to the beginning of the first line that you wish to move.**
- **Type `Ctrl-Space` to set the *mark*.**
- **Move the point to the end of the last line that you wish to move. Note that `emacs` highlights the *region* thus bounded by the point and the mark.**
- **Type `Ctrl-w` to *wipeout* the region. `emacs` moves the region to its clipboard.**
- **Move the point to some spot in the buffer**
- **Type `Ctrl-y` to yank (that is, copy) the text from the clipboard to the buffer at the point.**

(Note that the "minimal computing environment" described in our first precept is completely mouseless. To use the mouse (or a touch pad) with `emacs`, you can install an X Window System Server on your computer, as described in a forthcoming Piazza message.)

## Saving and Exiting

The `save-buffer` function (bound to `Ctrl-x Ctrl-s`) saves the buffer, that is, copies the contents of the buffer to its file on disk. **Type `Ctrl-x Ctrl-s` to save the buffer to the `testintmath.c` file.** As its name implies, the `save-buffers-kill-emacs` function (bound to `Ctrl-x Ctrl-c`) saves all `emacs` buffers to their respective files on disk, and exits `emacs`. (The section of this tutorial entitled *Managing Windows and Buffers* describes how you can use more than one `emacs` buffer simultaneously.) **Type `Ctrl-x Ctrl-c` to exit `emacs`, thus returning to the Bash prompt.**

## Indenting

At this point `testintmath.c` probably is seriously mangled. So **recopy the `testintmath.c` file from the `/u/cos217/emacs_testfiles` directory to your working directory. Then issue the command `emacs testintmath.c` to relaunch `emacs` to edit the `testintmath.c` file.**

`emacs` automatically indents C code as you type it, according to the indentation style that you specified in your `.emacs` file.

The `c-indent-command` function (bound to the `Tab` key) indents the current line according to the chosen indentation style. Note that the `Tab` key does not insert a tab character into your file; rather it

indents the current line. **Intentionally mal-indent a line, move the point to any spot within that line, and type the Tab key.**

The `indent-all` function (bound to `Ctrl-x p` because it indents your code *perfectly*) indents all lines of the buffer according to the chosen indentation style. **Intentionally mal-indent multiple lines scattered throughout the buffer, and then type `Ctrl-x p`.**

## Searching and Replacing

The `isearch-forward` function (bound to `Ctrl-s`) incrementally searches forward through the buffer for the text that you specify. This sequence illustrates:

- **Move the point to the beginning of the buffer.**
- **Type `Ctrl-s`, followed by the text "il"**
- **Type `Ctrl-s` repeatedly.**
- **Move the point, thereby ending the search.**

The similar `isearch-backward` function (bound to `Ctrl-r`) incrementally searches backward.

The `query-replace` function (bound to `Esc %`) incrementally replaces the "old" text that you specify with the "new" text that you specify. During execution of the function, typing `y` commands `emacs` to perform the replacement and continue executing the function, `n` commands `emacs` to skip the replacement and continue executing the function, `!` commands `emacs` to perform all replacements and stop executing the function, and `q` commands `emacs` to stop (*quit*) executing the function. For example:

- **Move the point to the beginning of the buffer**
- **Type `Esc %`, followed by "il", followed by "xxx".**
- **Type "y" and "n" a few times.**
- **Type "q".**
- **Move the point to the beginning of the buffer.**
- **Type `Esc %`, followed by "xxx", followed by "il".**
- **Type "!".**

## Managing Windows and Buffers

Recall that, in `emacs` jargon, a *buffer* is a region of memory, and a *window* is a graphical area which displays the contents of a buffer. So far in this tutorial you've used only one buffer and one window. More generally, at any given time, `emacs` will be managing multiple buffers and will be displaying some (but not necessarily all) of them in windows.

To *find* a file means to load it into a buffer. The `find-file` function (bound to `Ctrl-x Ctrl-f`) finds the file whose name you provide. **Type `Ctrl-x Ctrl-f hello.c` followed by the Enter key** to load the `hello.c` file into a buffer. Then **type `Ctrl-x Ctrl-f circle.c` followed by the Enter key** to load the `circle.c` file into a buffer. At this point `emacs` is managing three buffers; one of them is displayed in a window.

The `split-window-vertically` function (bound to `Ctrl-x 2`) splits the current window into two windows, each of which displays the same buffer. **Type `Ctrl-x 2` to split the current window into two windows.** The `other-window` function (bound to `Ctrl-x o`) moves the point to the other window. **Type `Ctrl-x o` a few times** to move the point back-and-forth between the two windows. Now **type `Ctrl-x Ctrl-f testintmath.c`** to find the `testintmath.c` file. At this point `emacs` is managing three buffers; two of them are displayed in `emacs` windows.

The `delete-other-window` function (bound to `Ctrl-x 1`) deletes the other window (that is, the window in which the point does not reside), thus returning `emacs` to its default one-window state. **Type `Ctrl-x 0` as necessary to move the point to the window that displays the `testintmath.c` buffer. Type `Ctrl-x 1` to delete the window that displays the `circle.c` buffer, leaving only the window that displays the `testintmath.c` buffer.** At this point `emacs` is managing three buffers; only one of them – the `testintmath.c` buffer – is displayed in a window.

With today's windowing operating systems, the ability of `emacs` to manage multiple windows is less important than it used to be. However, you must know about `emacs` windows to (1) use `gdb` within `emacs`, as will be described in an upcoming precept, and (2) build within `emacs`, as described in the next section of this tutorial.

## Building

Most COS 217 students build (that is, preprocess, compile, assemble, and link) C programs by issuing the `gcc217` command at the shell prompt. An alternative is to build C programs by issuing the `gcc217` command from within `emacs`. The alternative approach is optional in the COS 217 course.

The `compile` function (no keystroke binding) builds a C program from within `emacs` using whatever command you specify. This sequence illustrates:

- Intentionally introduce some compile-time errors into `testintmath.c`. Specifically, **change the return type of the `gcd()` function from `int` to `it`, and change the last line of the `gcd()` function from `return iFirst` to `retrn iFirst`.**
- **Type `Ctrl-x Ctrl-s` to save the `testintmath.c` buffer to disk.**
- **Type `Esc x compile`.** `emacs` assumes that you wish to use the `make -k` command to build. At this point in the course, that's incorrect. So **type the Backspace key repeatedly** to delete that command. **Then type:**  
**`gcc217 testintmath.c -o testintmath`.**
- **Type the Enter key.** `emacs` opens a *compilation* window, displaying error messages.
- **Type `Ctrl-x 0` to move the point to the compilation window.**
- **Move the point to one of the error messages, and type the Enter key.** `emacs` moves the point to the other window, to the offending line.
- **Correct the offending line.**
- **Use the same approach to correct the second offending line,** and thus build successfully.

## Miscellaneous Functions

The `undo` function (bound to `Ctrl-_`) undoes the previously executed function. **Move the point to some arbitrary spot in the buffer, type the Backspace key to delete a character, and then type `Ctrl-_` to undo that change.**

The `keyboard-quit` function (bound to `Ctrl-g`) aborts a multi-keystroke function call. **Type `Ctrl-x` to begin a keystroke sequence that calls a function; then type `Ctrl-g` to abort the function call. Type `Esc x` to begin a keystroke sequence that calls a function; then type `Ctrl-g` to abort the function call.**

The `linum` function (bound to `Ctrl-x n`) toggles the display of line numbers on the left side of the window. **Type `Ctrl-x n` to display line numbers; then type `Ctrl-x n` to undisplay them.**

**Type `Ctrl-x Ctrl-c` to save all buffers and exit `emacs`, thus ending the tutorial.**

## Part 2: Reference

This reference assumes that emacs is configured using the .emacs file provided to COS 217 students.

To type `Ctrl-somechar` (for any character *somechar*), type the *somechar* key while holding down the `Ctrl` key. To type `Esc somechar` (for any character *somechar*), type the `Esc` key followed by the *somechar* key.

On a PC using PuTTY, typing `Alt-somechar` has the same effect as typing `Esc somechar`. On a Mac that's true if and only if you have configured your Terminal application appropriately. To do that, from the Terminal application's menu choose Terminal | Preferences... | Settings | Keyboard and make sure the "Use option as meta key" checkbox is checked.

In emacs all work is accomplished by calling functions. To call a function, type `Esc x function`. Many functions are bound to keystrokes. Commonly used functions are in **boldface**.

### Moving the Point

Binding	Function	Description
<code>→</code>	<b>forward-char</b>	<b>Move the point forward one character</b>
<code>←</code>	<b>backward-char</b>	<b>Move the point backward one character</b>
<code>↓</code>	<b>next-line</b>	<b>Move the point to the next line</b>
<code>↑</code>	<b>previous-line</b>	<b>Move the point to the previous line</b>
<code>Ctrl-f</code>	forward-char	Move the point forward one character
<code>Ctrl-b</code>	backward-char	Move the point backward one character
<code>Ctrl-n</code>	next-line	Move the point to next line
<code>Ctrl-p</code>	previous-line	Move the point to previous line
<code>Esc f</code>	forward-word	Move the point to next word
<code>Esc b</code>	backward-word	Move the point to previous word
<b>Home</b>	<b>beginning-of-line</b>	<b>Move the point to beginning of line (but not with some terminal apps)</b>
<b>End</b>	<b>end-of-line</b>	<b>Move the point to end of line (but not with some terminal apps)</b>
<code>Ctrl-a</code>	<b>beginning-of-line</b>	<b>Move the point to beginning of line</b>
<code>Ctrl-e</code>	<b>end-of-line</b>	<b>Move the point to end of line</b>
<code>Esc a</code>	c-beginning-of-statement	Move the point to the beginning of C statement
<code>Esc e</code>	c-end-of-statement	Move the point to the end of C statement
<b>PageDn</b>	<b>scroll-up</b>	<b>Move the point to next page (but not with some terminal apps)</b>
<b>PageUp</b>	<b>scroll-down</b>	<b>Move the point to previous page (but not with some terminal apps)</b>
<code>Ctrl-v</code>	<b>scroll-up</b>	<b>Move the point to next page</b>
<code>Esc v</code>	<b>scroll-down</b>	<b>Move the point to previous page</b>
<code>Esc &lt;</code>	<b>beginning-of-buffer</b>	<b>Move the point to beginning of the buffer</b>
<code>Esc &gt;</code>	<b>end-of-buffer</b>	<b>Move the point to end of the buffer</b>
<code>Esc Ctrl-a</code>	beginning-of-defun	Move the point to beginning of the C function
<code>Esc Ctrl-e</code>	end-of-defun	Move the point to end of the C function
<code>Ctrl-x l line</code>	<b>goto-line</b>	<b>Move the point to line whose number is <i>line</i></b>

### Inserting and Deleting

Binding	Function	Description
<b>Bsp</b>	<b>c-electric-backspace</b>	<b>Delete the character before the point</b>
<code>Esc Bsp</code>	backward-kill-word	Delete the characters from the point to the beginning of the word
<code>Ctrl-d</code>	<b>c-electric-delete-forward</b>	<b>Delete the character at the point</b>
<code>Ctrl-k</code>	<b>kill-line</b>	<b>Cut the current line</b>
<code>Ctrl-Sp</code>	<b>set-mark-command</b>	<b>Set the mark at the point</b>
<code>Ctrl-x Ctrl-x</code>	exchange-point-and-mark	Exchange the mark and the point
<code>Ctrl-x h</code>	mark-whole-buffer	Set the point at the beginning and the mark at the end of the buffer
<code>Ctrl-w</code>	<b>kill-region</b>	<b>Cut the region denoted by the mark and the point</b>
<code>Esc w</code>	kill-ring-save	Copy the region denoted by the mark and the point
<code>Ctrl-y</code>	<b>yank</b>	<b>Paste the previously cut/copied region at the point</b>

### Saving and Exiting

Binding	Function	Description
<code>Ctrl-x Ctrl-s</code>	<b>save-buffer</b>	<b>Save the current buffer to its file</b>
<code>Ctrl-x Ctrl-w file</code>	write-file	Write the current buffer to <i>file</i>
<code>Ctrl-x Ctrl-q</code>	vc-toggle-read-only	Toggle the current buffer between read-only and read/write
<code>Ctrl-x Ctrl-c</code>	<b>save-buffers-kill-emacs</b>	<b>Save all buffers and exit Emacs</b>

## Indenting

Binding	Function	Description
Ctrl-c .	c-set-style	Set the C indentation style to the specified one
<b>TAB</b>	<b>c-indent-command</b>	<b>Indent the current line of the C program</b>
Esc Ctrl-\	indent-region	Indent the region of the C program denoted by the mark and the point
<b>Ctrl-x p</b>	<b>indent-all</b>	<b>Indent all lines of the C program (i.e. indent the program perfectly)</b>

## Searching and Replacing

Binding	Function	Description
<b>Ctrl-s <i>string</i></b>	<b>isearch-forward</b>	<b>Search forward for <i>string</i></b>
Ctrl-r <i>string</i>	isearch-backward	Search backward for <i>string</i>
Esc % <i>old new</i>	query-replace	Replace the <i>old</i> string with the <i>new</i> one y => replace n => skip ! => replace all q => quit

## Managing Windows and Buffers

Binding	Function	Description
<b>Ctrl-x Ctrl-f <i>file</i></b>	<b>find-file</b>	<b>Load <i>file</i> into a buffer</b>
Ctrl-x Ctrl-r <i>file</i>	find-file-read-only	Load <i>file</i> into a buffer for read only
<b>Ctrl-x 2</b>	<b>split-window-vertically</b>	<b>Split the current window into two windows arranged vertically</b>
<b>Ctrl-x o</b>	<b>other-window</b>	<b>Move the point to the other window</b>
Ctrl-x 3	split-window-horizontally	Split the current window into two windows arranged horizontally
Ctrl-x 0	delete-window	"Undisplay" the current window
<b>Ctrl-x 1</b>	<b>delete-other-windows</b>	<b>"Undisplay" all windows except the current one</b>
Ctrl-x Ctrl-b	list-buffers	Display a new window listing all buffers
Ctrl-x b <i>file</i>	switch-to-buffer	Load <i>file</i> into a buffer if necessary, and then display that buffer in the current window

## Building and Debugging

Binding	Function	Description
	<b>compile <i>command</i></b>	<b>Build the program using <i>command</i></b>
	<b>gdb <i>executablefile</i></b>	<b>Launch the GDB debugger to debug <i>executablefile</i></b>

## Miscellaneous

Binding	Function	Description
Ctrl-x u	undo	Undo the previous change
<b>Ctrl-<u></u></b>	<b>undo</b>	<b>Undo the previous change</b>
<b>Ctrl-g</b>	<b>keyboard-quit</b>	<b>Abort the multi-keystroke command</b>
Ctrl-h	help-command	Access the Emacs help system
Esc `	tmm-menubar	Access the Emacs menu
<b>Ctrl-x n</b>	<b>linum</b>	<b>Display/undisplay a line number before each line</b>